Elixir - GenStage

Mikołaj Tomalik
Piotr Matląg

Czym jest GenStage?

W oficjalnej dokumentacji możemy przeczytać, że jest to „specyfikacja i przepływ obliczeniowy dla Elixira”.

W praktyce oznacza to, że GenStage umożliwia nam zdefiniowanie potoku prac, które mają być wykonane przez niezależne etapy w oddzielnych procesach.

[A] -> [B] -> [C]

```elixir
defmodule A do
 use GenStage
 def init(counter) do
   {:producer, counter}
 end
 def handle_demand(demand, counter) when demand > 0 do
   events = Enum.to_list(counter..counter+demand-1)
   {:noreply, events, counter + demand}
 end
end
```

```elixir
defmodule B do
 use GenStage
 def init(number) do
   {:producer_consumer, number}
 end
 def handle_events(events, _from, number) do
   events = for event <- events,
         entry <- [{event, event * 2}],
         do: entry
   {:noreply, events, number}
 end
end
```

```elixir
defmodule C do
 use GenStage
 def init(:ok) do
   {:consumer, :the_state_does_not_matter}
 end
 def handle_events(events, _from, state) do
   :timer.sleep(5000)
   IO.inspect(events)
   {:noreply, [], state}
 end
end
```

```elixir
{:ok, a} = GenStage.start_link(A, 0)

{:ok, b} = GenStage.start_link(B, 2)

{:ok, c} = GenStage.start_link(C, :ok)


GenStage.sync_subscribe(b, to: a)

GenStage.sync_subscribe(c, to: b)
```

W skrócie

A - tworzy tyle liczb całkowitych, ile jest potrzebne

B - przekształca otrzymane dane na krotki

C - wypisuje otrzymane dane do terminala

[A] -> [B] -> [C]

Producer

```
defmodule A do
  use GenStage

  def init(counter) do
    {:producer, counter}
  end

  def handle_demand(demand, counter) when demand > 0 do
    events = Enum.to_list(counter..counter+demand-1)
    {:noreply, events, counter + demand}
  end
end
```

Consumer

```
defmodule C do
  use GenStage

  def init(:ok) do
    {:consumer, :the_state_does_not_matter}
  end

  def handle_events(events, _from, state) do
    :timer.sleep(2000)

    IO.inspect(events)

    {:noreply, [], state}
  end
end
```

Consumer-Producer

```
defmodule B do
  use GenStage

  def init(number) do
    {:producer_consumer, number}
  end

  def handle_events(events, _from, number) do
    events =
      for event <- events,
          entry <- [{event, event * number}],
          do: entry
    {:noreply, events, number}
  end
end
```

Connection

```
{:ok, a} = GenStage.start_link(A, 0)
{:ok, b} = GenStage.start_link(B, 5)
{:ok, c} = GenStage.start_link(C, :ok)

GenStage.sync_subscribe(b, to: a)
GenStage.sync_subscribe(c, to: b)
```

Memy!

Realny przykład zastosowania